A Proposal for Publishing Data Streams as Linked Data
- A Position Paper -

Davide F. Barbieri
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza L. da Vinci 32, 20133 Milano
dbarbieri@elet.polimi.it

Emanuele Della Valle
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza L. da Vinci 32, 20133 Milano
dellavalle@elet.polimi.it

ABSTRACT
Streams are appearing more and more often on the Web in sites that distribute and present information in real-time streams. We anticipate a rapidly growing need of mashing up this streaming information with more static one. While best practices for linking static data on the Web were published and facilitate the mash up of static information published on the Web, streams were neglected. In this short position paper, we propose an approach to publish Data Streams as Linked Data.

Keywords
Data Streams, Linked Data, Virtual RDF, Stream Reasoning

1. INTRODUCTION
A growing number of Web sites are distributing and presenting information in real-time streams. Microblogs such as Twitter\(^1\), weather monitoring sites such as AccuWeather\(^2\), traffic monitoring sites such as Waze\(^3\) are few representative examples. Streams, being unbounded sequences of time-varying data elements, should not be treated as persistent data to be stored (forever) and queried on demand, but rather as transient data to be consumed on the fly by continuous queries. Continuous queries, after being registered, keep analyzing such streams, producing answers triggered by the streaming data and not by explicit invocation. Such a paradigmatic change have been largely investigated in the last decade by the database community \[15\]. Specialized Data Stream Management Systems (DSMS) have been developed (e.g., STREAM\(^2\), Aurora/Borealis \[1\] and Stream Mill \[6\]). Several startups such as StreamBase\(^4\) are commercializing DSMS, and features of DSMS are becoming supported by major database products, such as Oracle and DB2.

Motivated by the availability of real-time streams on the Web and by the lack of Web-based approaches to process them, we have been working since 2008 on an extension to SPARQL\[^{20}\] for continuous querying over streams of RDF and static RDF graphs (namely C-SPARQL \[7, 9\]).

Listing 1 shows an example of C-SPARQL which allows dealing with streams of RDF triples as well as static RDF graphs

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1    | REGISTER STREAM TotalAmountPerBroker COMPUTE EVERY 10m AS | \[
| 2    | PREFIX ex: <http://example/> | \[
| 3    | FROM STREAM <http://stockex.org/market.trdf> | \[
| 4    | WHERE { | \[
| 5    | ?broker ex:hasTotalAmount ?total . } | \[
| 6    | FILTER (?total = ?amount) | \[
| 7    | CONSTRUCT {?broker ex:hasTotalAmount ?total .} | \[
| 8    | AGGREGATE { (?total, SUM(?amount), ?broker) } | \[
| 9    | FROM <http://brokerscentral.org/brokers.rdf> | \[
| 10   | } | \[
| 11   | \]
| 12   | \]

At line 1, the `REGISTER` clause is used to tell the C-SPARQL engine that it should register a continuous query, i.e., a query that will continuously compute answers to the query. In particular, we are registering a query that generates an RDF stream. The `COMPUTE EVERY` clause states the frequency of every new computation, in the example every 10 minutes. At line 5, the clause `FROM STREAM` defines the RDF stream of financial transactions, used within the query. Next, line 6 defines the `window` of observation of the RDF stream. Streams, for their very nature, are volatile and for this reason should be consumed on the fly; thus, they are observed through a window, including the last elements of the stream, which changes over time. In the example, the window comprises RDF triples produced in the last 1 hour, and the window slides every 10 minutes. The `WHERE` clause is standard; it includes a set of matching patterns and filters clauses as in standard SPARQL. Finally, at line 13, the `AGGREGATE` function asks the C-SPARQL engine to include in the result set a new variable `?total` which is bound to the sum of the amount of the transaction of each broker.

Our C-SPARQL Engine \[^{9}\] treats non-RDF DSMSs as virtual RDF streams and graphs. It allows to register queries that continuously combine (virtual) RDF streams and RDF graphs. Under this respect, our C-SPARQL Engine is similar to D2RQ \[^{12}\] that treats non-RDF databases as virtual RDF graphs. In our previous works \[7, 8, 9\] we develop an engine for registering and continuously executing C-SPARQL queries. With this position paper, we propose an extension of our C-SPARQL Engine that publishes data streams as Linked Data. Such an extension complements the work done so far and lowers the entry barrier for external applications.

\(^1\)http://twitter.com/\n\(^2\)http://world.waze.com/\n\(^3\)http://www.accuweather.com/\n\(^4\)http://www.streambase.com/

Copyright is held by the author(s).
3. STREAM DATA MANAGEMENT FOR THE SEMANTIC WEB

(Semantic) Web applications to consume data streams.

The rest of the paper is organized as follows. In Section 2, we describe the design principles that inspire our proposal for Streaming Linked Data. Section 3 explains how to publish a single data stream as an RDF stream. In the same section, we also present a vocabulary to describe the time interval in which the published data are valid. The URI schema that allows to control the Window behavior is presented in Section 4. In Section 5, we describe the RESTful [21] services which allow to control the C-SPARQL query that continuously computes the published RDF stream. Finally, Section 6 and 7 present some related work and draw some conclusions, respectively.

2. DESIGN PRINCIPLE

The design principle that inspires our approach is illustrated in Figure 1. Our C-SPARQL engine is able to process data streams and RDF streams in combination with RDF graphs. In our previous work, we use in memory connection between our C-SPARQL engine and local C-SPARQL clients. However, we anticipate a rapidly growing need of clients. However, we anticipate a rapidly growing need of services that implement a remote C-SPARQL Client (see Section 5). Such services provide full control (i.e., beyond window behavior) on the C-SPARQL queries whose results are served as Linked Data by the Streaming Linked Data Server.

3. PUBLISHING A STREAM

A data stream is defined as an ordered sequence of pairs, where each pair is made of a tuple and its timestamp $\tau$. For instance, the stream of financial transactions used in the example in Listing 1 could contain a transaction $tr1$ done by broker1 for $1000$ registered at $\tau_1$, and two transactions at $\tau_{i+1}$: $tr2$ done by broker1 for $3000$ and $tr3$ done by broker2 for $2000$.

Listing 1: Example of Stream Graph linking two Instantaneous Graphs

```sparql
PREFIX : <http://example/> .

:sgraph1 sld:hasSource :igraph1 .
:sgraph1 sld:hasSource :igraph2 .
:sgraph1 sld:lastUpdate "2015-01-31T12:00:00Z"^^xsd:dateTime .
:sgraph1 sld:windowType sld:logicalTumbling .
:sgraph1 sld:windowSize "1 hour" .
:tr1 :with "$1000" .
:tr2 :with "$2000" .
:tr3 :with "$3000" .

:tr1 sld:transactedAt "2015-01-31T12:01:00Z"^^xsd:dateTime .
:tr1 sld:broker "broker1" .
:tr1 sld:amount "$1000" .

:tr2 sld:transactedAt "2015-01-31T12:02:00Z"^^xsd:dateTime .
:tr2 sld:broker "broker1" .
:tr2 sld:amount "$2000" .

:tr3 sld:transactedAt "2015-01-31T12:03:00Z"^^xsd:dateTime .
:tr3 sld:broker "broker2" .
:tr3 sld:amount "$3000" .
```

We propose to represent RDF streams in RDF using named graphs [13]. We distinguish between two kinds of named graphs: the Stream Graphs (shortly s-graphs) and the Instantaneous Graphs (shortly i-graphs). In our proposal, an RDF Stream can be represented using one s-graph and several i-graphs, one for each timestamp.

A s-graph is a metadata graph that describes the current content of the window over the RDF Stream. The most important part of an s-graph are the triples that refer to the i-graphs using rdfs:seeAlso3 and those that describe when each i-graph was received using the property receivedAt.

Few other metadata complete the description of an s-graph. The property lastUpdate describes the last time the graph was updated. The property expires allows to indicate a Linked Data Client that the information in the graph will expire in a given moment in future. The properties sld:windowType and sld:windowSize describe the window through which the stream is observed (see Section 4 for more information).

For instance, if the data stream exemplified above was the current content of a window over the stream of financial transactions, it can be represented using the s-graph in Listing 2 and the two i-graphs in Listing 3 and 4.

Listing 2: Example of Stream Graph linking two Instantaneous Graphs

```sparql
PREFIX : <http://example/> .

:sgraph1 sld:hasSource :igraph1 .
:sgraph1 sld:hasSource :igraph2 .
:sgraph1 sld:lastUpdate "2015-01-31T12:00:00Z"^^xsd:dateTime .
:sgraph1 sld:windowType sld:logicalTumbling .
:sgraph1 sld:windowSize "1 hour" .
:tr1 :with "$2000" .

:tr1 sld:transactedAt "2015-01-31T12:01:00Z"^^xsd:dateTime .
:tr1 sld:broker "broker1" .
:tr1 sld:amount "$2000" .

:tr2 sld:transactedAt "2015-01-31T12:02:00Z"^^xsd:dateTime .
:tr2 sld:broker "broker1" .
:tr2 sld:amount "$1000" .

:tr3 sld:transactedAt "2015-01-31T12:03:00Z"^^xsd:dateTime .
:tr3 sld:broker "broker2" .
:tr3 sld:amount "$3000" .
```

Listing 3: The Instantaneous Graph timestamped with $\tau_i$.

3. We choose to link s-graphs to i-graphs using the property rdfs:seeAlso, because it has been largely adopted to link named graphs (see for instance the usage of rdfs:seeAlso in Sindice [19] and in the Semantic Web Client [17]).
As we have explained in the previous section, streams are intrinsically infinite. In C-SPARQL, we introduce the notion of windows over streams. In Section 3, we focus on the general approach to publish a data stream rather than on the notion of window. However, we foresee the need for a consumer of Streaming Linked Data to be able to control the behavior of the window through which the stream is observed.

Types and characteristics of windows in C-SPARQL are inspired by those of the windows defined in continuous query languages for relational streaming data, such as CQL[3].

Windows are expressed in C-SPARQL within the FROM clause, whose syntax is as follows:

```
FROM StrClause AS ([RANGE] [STREAM] StreamIRI |
[ [RANGE] Window ] |
 LogicalWindow | Number TimeUnit WindowOverlap |
 TimeUnit | 's' | 'm' | 'h' | 'd' | 'w' | 'y' | 'a' |
 WindowOverlap | TO | Number TimeUnit | [TUMBLING] |
 PhysicalWindow → TRIPLES | Number |
```

A window extracts from the stream the last data stream elements, which are considered by the query. Such extraction can be physical (a given number of triples) or logical (all the triples which occur during a given time interval, the number of which is variable over time).

Logical windows are sliding [16] when they are progressively advanced of a given step (i.e. a time interval that is shorter than the window’s time interval); they are non-overlapping (or tumbling) when they are advanced of exactly their time interval at each iteration. With tumbling windows every triple of the stream is included exactly into one window, whereas with sliding windows some triples can be included into several windows.

We believe that consumers of Streaming Linked Data would largely benefit from controlling the window of a running C-SPARQL query. Therefore we propose the following URI schemata:

- physical windows can be controlled replacing \[Size\] with the number of triples (e.g., the last 1000 triples)

  ```
  "http://ex.org/transactions/physical/1000"
  ```

- logical windows can be controlled replacing \[Size\] with a time interval (e.g., PT1H meaning 1 hour) and replacing \[Range\] either with the keyword tumbling or with a time interval (e.g., PT10M meaning 10 minutes).

  ```
  "http://ex.org/transactions/logical/PT1H/tumbling"
  ```

Notably, each of these IRIs are translated to an equivalent C-SPARQL query that processes the data stream. For instance, the example above is equivalent to the following C-SPARQL query:

```
PREFIX : <http://example/>
PREFIX sld: <http://www.streaminglinkeddata.org/schema#>

CONSTRUCT *
WHERE { ?s ?p ?o . }

@prefix : <http://example/>.
@prefix sld: <http://www.streaminglinkeddata.org/schema#>.
```

4. CONTROLLING THE WINDOW

4.1 Physical windows

Physical windows are expressed in C-SPARQL within the PhysicalWindow.

```
PhysicalWindow → TRIPLES | Number |
```

A physical window extracts from the stream the last data stream elements, which are considered by the query.

5. CONTROLLING C-SPARQL QUERIES

5.1 GET

```
GET /transactions/physical/1000 
```

5.2 DELETE

```
DELETE /transactions/physical/1000
```

5.3 PUT

```
PUT /transactions/physical/1000
```

6. RELATED WORK
3. STREAM DATA MANAGEMENT FOR THE SEMANTIC WEB

At a first glance, his proposal could appear similar to ours. Both his and our proposal use named graphs and define IRI schemata. However, his approach does not take into account the nature of streams, that, being unbounded sequences of time-varying data elements, should not be treated as persistent data to be stored (forever) and queried on demand, but rather as transient data to be consumed on the fly by continuous queries. His proposal allows for opening a window starting from and ending into any moment in time (see listing below). This is incompatible with the principle to keep a window open on the latest data that has to be consumed on the fly. It requires the Linked Stream Data server to store the stream for an indefinite time period.

http://www.dmw-project.org/excerpt/mem.txt/start time%,end time%

In [22], Rodríguez et al. introduce the notion of Time-Annotated RDF (TA-RDF) that allows for representing time-series data, especially streaming data, using the Semantic Web approach. TA-RDF is an extension of the RDF model where resources are optionally annotated with a time value, i.e., a time-annotated resource is a pair of the form resource[time] (see listing below for an example).

A TA-RDF graph can be represented as a set of RDF graphs using two special properties: beingTo, which indicates a data element in a stream, and hasTimestamp, which points toward the timestamp of the data element. As for the previous related work, TA-RDF proposal looks very similar to ours, but still it lacks the paradigmatic change from persistent data to transient data. In TA-RDF streams are supposed to be stored indefinitely.

Finally, the two proposals do not consider the rich types of windows proposed in DSMS. They do not propose a vocabulary to describe the window type (i.e., lsd:physical vs. lsd:logical) and the size of the window (i.e., the equivalent of our property windowSize). The properties lastUpdate and expires, which in our vocabulary allows to indicate a Linked Data Client when the graph was updated and when it will expire, are not present.

7. CONCLUSION

Distributing and presenting information in real-time streams is becoming a best practice on the Web. The nature of streams requires a paradigmatic change from persistent data to be stored, and queried on demand, to transient data, to be consumed on the fly by continuous queries.

In our previous work we investigated C-SPARQL as an approach to treat non-RDF DSMSs as virtual RDF streams and graphs. With this position paper, we propose an extension of our C-SPARQL Engine that publishes data streams as Linked Data. In this paper, we described the principle that inspires our approach and we explain how to publish RDF streams continuously generated by C-SPARQL queries. Such a best practice introduces the concepts of Stream Graph (or s-graph) and Instantaneous Graph (or i-graph) as well as a small vocabulary that allows to describe which part of the stream has been published and when the information will expire. A RESTful service to control the C-SPARQL queries that generates the RDF streams is also detailed.

We believe that our proposal can lower the entry barrier for external (Semantic) Web applications to consume data streams. Our next step is to complete the prototypical implementation of our Streaming Linked Data Server and evaluate it against several use cases. We are currently considering the synthetic Linear Road Benchmark [4], a well-established benchmark for Data Stream Management Systems, and several real source of streams that we are already experimenting with (see for instance, the social media streams in [8] or the Milan traffic streams in [9]).

8. ACKNOWLEDGMENTS

The work described in this paper has been partially supported by the European project LarKC (FP7-215535).

9. REFERENCES


